Module 1: Repository Structure in Scientific Julia Packages

Madhusudhan Pandey, PhD

November 21, 2025

Abstract

This lecture provides an overview of the standard repository structure used in modern scientific Julia packages, using ModelingToolkit.jl as a representative example. The session introduces graduate and PhD students to the essential folders, configuration files, and automation tools that support high-quality research software development. Key concepts include documentation workflows, benchmarking, modular extensions, testing infrastructures, and continuous integration pipelines. By understanding the purpose and interplay of components such as `src`, `docs`, `test`, and the `.github` automation ecosystem, students gain the foundational skills required to design, maintain, and publish professional scientific software. The lecture equips learners to build reproducible, scalable, and maintainable research codebases aligned with best practices in computational science and engineering.

This lecture has several modules. **This is Lecture Module 1**. Each module in this lecture series is designed as a self-contained 30-minute session. The material may be taken independently or with an instructor, and is suitable for learners at the beginner, intermediate, or advanced level. This flexibility allows students to progress at their own pace while ensuring that each module provides a complete and accessible learning experience.

1 Repository Structure Overview

This lecture introduces the standard repository organisation used in modern scientific Julia projects.[1] We use `ModelingToolkit.jl` as a canonical example from the SciML ecosystem. The items are listed in their natural, top–down ordering within a GitHub repository. Figure 1 shows a repository structure of *ModelingToolkit.jl* on Github.

1.1 Folders

1.1.1 `.github/`

Contains GitHub automation tools including CI/CD workflows, issue templates, and pull request templates. **Concept:** Automate testing, documentation building, formatting checks, and enforce quality assurance across contributions.

1.1.2 `benchmark/`

Includes benchmarking scripts that measure the performance of critical package features across releases. **Concept:** Maintain computational efficiency for research workloads and avoid regressions.

1.1.3 `docs/`

The full documentation infrastructure built using `Documenter.jl`. Contains tutorials, manuals, examples, and deployment configuration. **Concept:** Make documentation a first-class citizen to support scientific reproducibility.

1.1.4 `ext/`

Extension modules enabled only when optional dependencies are present. **Concept:** Modular design that keeps the base package lightweight while enabling advanced add-ons.

[1] This section contains AI-generated explanatory text (AI slop).

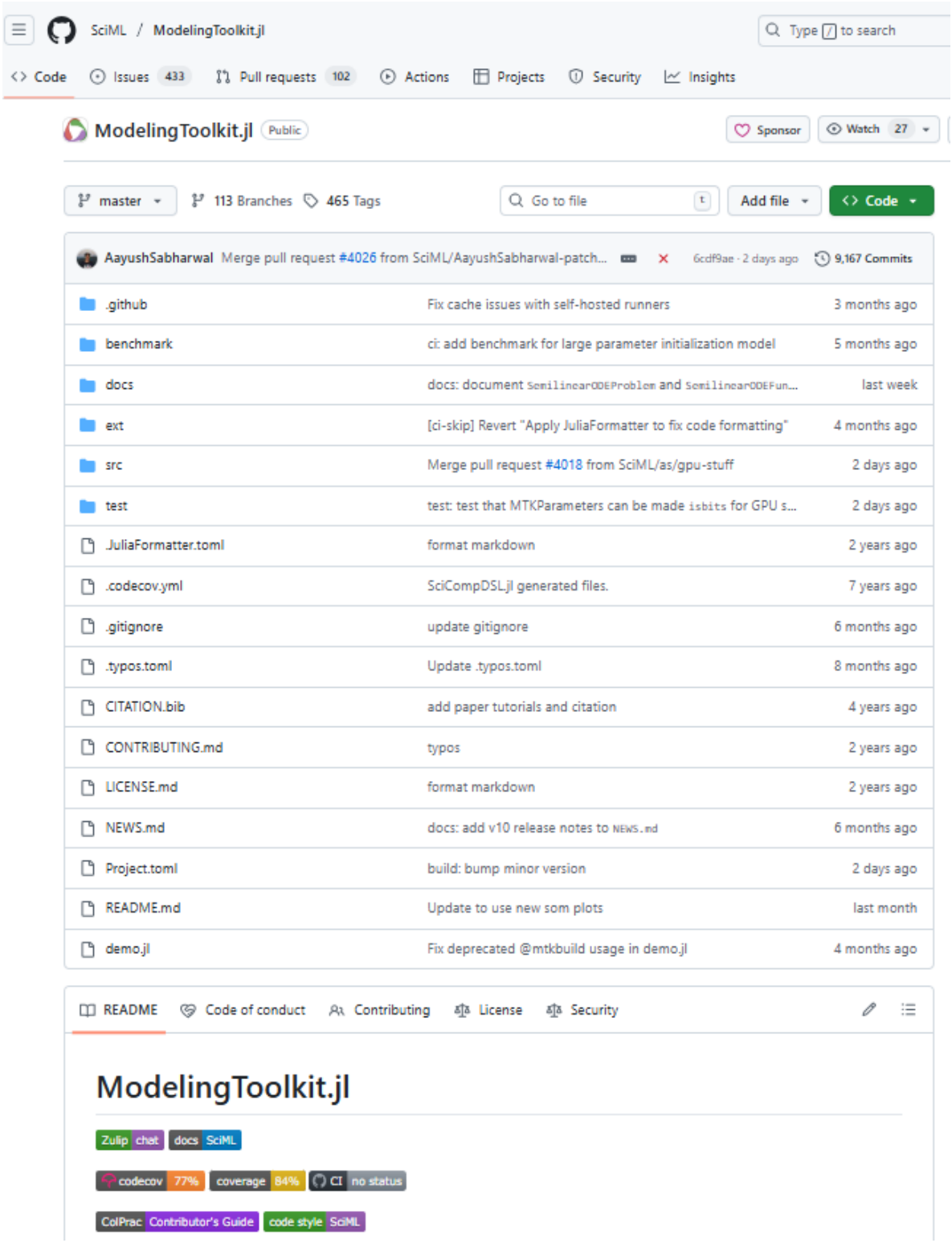

Figure 1: Repository structure of `ModelingToolkit.jl` on GitHub.

1.1.5 `src/`

The core implementation of the package, containing:

- module definitions

- types and structs
- symbolic systems
- differential-algebraic formulations
- algorithms and utilities

Concept: The computational "engine room" of the codebase.

1.1.6 `test/`

Contains unit tests executed through `Pkg.test`. Ensures correctness, prevents regressions, and validates mathematical behaviour. **Concept:** Provide scientific reliability and reproducibility via automated testing.

1.2 Top-Level Files

1.2.1 `.JuliaFormatter.toml`

Formatting rules for automatic code style enforcement. **Concept:** Maintain consistent coding style across collaborators.

1.2.2 `.codecov.yml`

Configuration for uploading coverage results to Codecov. **Concept:** Track how much of the codebase is tested.

1.2.3 `.gitignore`

Specifies files to exclude from version control (logs, artefacts, build outputs). **Concept:** Keep the repository clean and efficient.

1.2.4 `.typos.toml`

Configuration for automated spell checking across source and documentation. **Concept:** Improve documentation and code professionalism.

1.2.5 `CITATION.bib`

BibTEX entry enabling academic citation of the software. **Concept:** Support open, citable scientific software.

1.2.6 `CONTRIBUTING.md`

Instructions for contributors regarding code style, workflows, and testing. **Concept:** Standardise contributions for sustainable research development.

1.2.7 `LICENSE.md`

Specifies the legal software license. **Concept:** Clarify rights for use, sharing, modification, and distribution.

1.2.8 `NEWS.md`

Version-by-version change log. **Concept:** Transparent tracking of software evolution.

1.2.9 `Project.toml`

Defines:

- package name and UUID
- version
- dependencies
- compatibility rules

Concept: Declare the project as a Julia package.

1.2.10 `README.md`

Introductory documentation shown on the GitHub front page. **Concept:** Provide a quick start and essential overview.

1.2.11 `demo.jl`

A runnable example script demonstrating usage. **Concept:** Practical starting point for new users.

2 Summary Table

Item	Concept / Purpose
`.github/`	CI/CD workflows and automation
`benchmark/`	Performance measurement
`docs/`	Documentation infrastructure
`ext/`	Optional extension modules
`src/`	Core implementation
`test/`	Automated testing
`.JuliaFormatter.toml`	Code formatting rules
`.codecov.yml`	Coverage reporting
`.gitignore`	Ignore unnecessary files
`.typos.toml`	Spell-check configuration
`CITATION.bib`	Academic citation
`CONTRIBUTING.md`	Contributor guidelines
`LICENSE.md`	Legal licensing
`NEWS.md`	Version changes
`Project.toml`	Package identity
`README.md`	Usage overview
`demo.jl`	Example usage

3 GitHub Automation Workflows

The `.github/workflows` directory contains GitHub Actions automation scripts. Each file defines a continuous integration (CI) or maintenance job that runs automatically in response to events such as pushes, pull requests, tagging releases, or scheduled intervals. The following provides a concise conceptual overview of each workflow file.

3.1 Workflow Files in `.github/workflows/`

3.1.1 `CompatHelper.yml`

Runs the CompatHelper bot, which automatically opens pull requests to update version bounds for dependencies in `Project.toml`. **Purpose:** Ensure the package stays compatible with the latest ecosystem without breaking constraints.

3.1.2 `Documentation.yml`

Builds and deploys the package documentation using `Documenter.jl`. **Purpose:** Keep online documentation updated whenever new code is merged.

3.1.3 `Downgrade.yml`

Tests the package against older versions of dependencies or Julia. **Purpose:** Guarantee backward compatibility across environments.

3.1.4 `Downstream.yml`

Runs tests for external packages that depend on this package. **Purpose:** Detect whether changes break downstream users.

3.1.5 `FormatCheck.yml`

Runs JuliaFormatter to ensure consistent code formatting and rejects unformatted pull requests. **Purpose:** Enforce style uniformity across contributors.

3.1.6 `ReleaseTest.yml`

Runs additional tests or checks that should occur only during a tagged release. **Purpose:** Validate correctness before publishing a new version.

3.1.7 `SpellCheck.yml`

Checks for spelling mistakes in documentation, comments, and docstrings. **Purpose:** Maintain textual quality and professionalism.

3.1.8 `TagBot.yml`

Automatically creates GitHub releases and tags once a version is registered in the Julia General registry. **Purpose:** Automate release management and version tagging.

3.1.9 `Tests.yml`

The main CI workflow that runs `Pkg.test()` on multiple Julia versions and operating systems. **Purpose:** Ensure correctness, stability, and reproducibility of the package across platforms.

3.1.10 `benchmark.yml`

Runs performance benchmarks on scheduled intervals or on request. **Purpose:** Track performance regressions or improvements over time.

3.2 Other GitHub Automation Files

3.2.1 `dependabot.yml`

Configures Dependabot to monitor external dependencies and automatically open pull requests when updates become available. **Purpose:** Keep dependencies secure and up to date.

4 Summary Table: GitHub Workflow Automation

5 First two github automation Workflow Files in `ModelingToolkit.jl`

5.1 `CompatHelper.yml`: Dependency Compatibility Automation

The `CompatHelper.yml` workflow is executed daily via a cron schedule. It ensures that the package remains compatible with the latest versions of its dependencies.

Workflow File	Purpose / Concept
`CompatHelper.yml`	Automatically update version bounds
`Documentation.yml`	Build and deploy documentation
`Downgrade.yml`	Test compatibility with older versions
`Downstream.yml`	Test dependent downstream packages
`FormatCheck.yml`	Enforce automatic code formatting
`ReleaseTest.yml`	Run release-only pre-publication tests
`SpellCheck.yml`	Spell-check documentation and comments
`TagBot.yml`	Generate tags/releases automatically
`Tests.yml`	Main test suite across Julia/OS versions
`benchmark.yml`	Run performance benchmarks
`dependabot.yml`	Auto-update dependencies

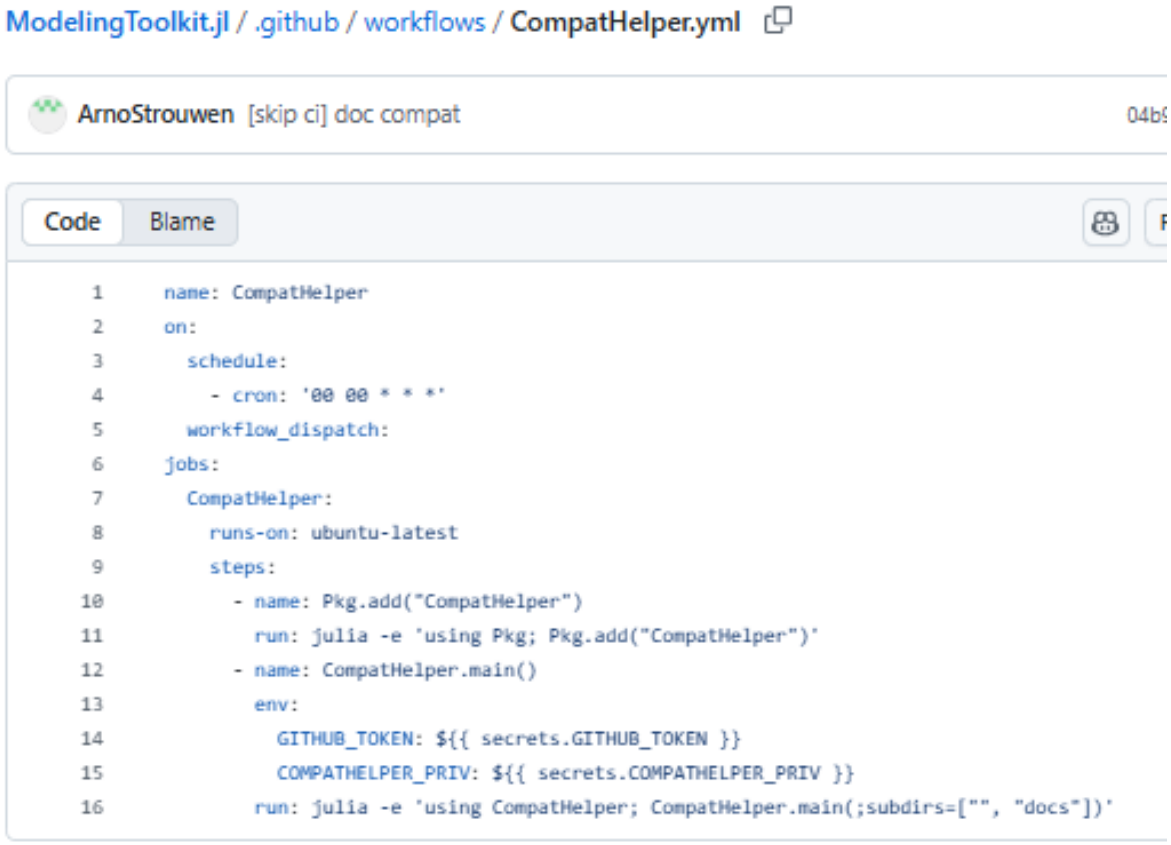

Figure 2: Repository structure of `CompatHelper.yml` from `.github/workflows/` folder of `ModelingToolkit.jl` on GitHub.

Conceptual Summary

- Installs the `CompatHelper` Julia package.
- Uses the GitHub token and a private RSA key to authenticate.
- Automatically opens pull requests to update `Project.toml`.

Purpose: Reduce maintenance burden and ensure ecosystem-wide compatibility for fast-moving scientific libraries.

ModelingToolkit.jl / .github / workflows / Documentation.yml

View Runs

AayushSabharwal ci: run workflows on PR to v10 branch 08075ce · 6 months ago History

Code Blame Raw

```yaml
name: Documentation

on:
  push:
    branches:
      - master
      - v10
    tags: '*'
  pull_request:

concurrency:
  # Skip intermediate builds: always, but for the master branch.
  # Cancel intermediate builds: always, but for the master branch.
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: ${{ github.ref != 'refs/heads/master' && github.refs != 'refs/tags/*'}}

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: julia-actions/setup-julia@latest
        with:
          version: 'lts'
      - run: sudo apt-get update && sudo apt-get install -y xorg-dev mesa-utils xvfb libgl1 freeglut3-dev libxrandr
      - name: Install dependencies
        run: DISPLAY=:0 xvfb-run -s '-screen 0 1024x768x24' julia --project=docs/ -e 'using Pkg; Pkg.develop(Packag
      - name: Build and deploy
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # For authentication with GitHub Actions token
          DOCUMENTER_KEY: ${{ secrets.DOCUMENTER_KEY }} # For authentication with SSH deploy key
          JULIA_DEBUG: "Documenter"
        run: DISPLAY=:0 xvfb-run -s '-screen 0 1024x768x24' julia --project=docs/ --code-coverage=user docs/make.jl
      - uses: julia-actions/julia-processcoverage@v1
      - uses: codecov/codecov-action@v5
        with:
          files: lcov.info
          token: ${{ secrets.CODECOV_TOKEN }}
          fail_ci_if_error: true
```

Figure 3: Repository structure of `Documentation.yml` from `.github/workflows/` folder of `ModelingToolkit.jl` on GitHub.

5.2 `Documentation.yml`: Automated Documentation Build

The `Documentation.yml` workflow defines the full automation pipeline for building and deploying package documentation. It triggers on pushes to the `master` and `v10` branches, as well as on tagged releases.

Conceptual Summary

- Installs Julia and required system dependencies.
- Runs documentation builds in a headless X virtual framebuffer (XVFB).

- Uses `Documenter.jl` with SSH authentication to deploy docs.
- Uploads documentation code coverage to Codecov.

Purpose: Guarantee that documentation is always consistent with the latest code and is automatically deployed without manual intervention.